

华中科技大学

2018

计算机组成原理

课程设计报告

题 目： 5 段流水 CPU 设计

专 业： 计算机科学与技术

班 级： 计卓 1501

学 号： U201516994

姓 名： 李沐辰

电 话： 15827045386

邮 件： 779369400@qq.com

完成日期： 2018-4-2



计算机科学与技术学院

华中科技大学课程设计报告

目 录

1	课程设计概述.....	3
1.1	课设目的	3
1.2	设计任务	3
1.3	设计要求	3
1.4	技术指标	4
2	总体方案设计.....	6
2.1	单周期 CPU 设计	6
2.2	中断机制设计.....	7
2.3	流水 CPU 设计.....	10
2.4	气泡式流水线设计.....	12
2.5	数据转发流水线设计	14
3	详细设计与实现.....	16
3.1	单周期 CPU 实现	16
3.2	中断机制实现.....	21
3.3	流水 CPU 实现	24
3.4	气泡式流水线实现.....	26
3.5	数据转发流水线实现	28
3.6	重定向流水线上板实现	30
4	实验过程与调试.....	31
4.1	测试用例和功能测试.....	31
4.2	课设思考题与主要问题的记录	32
4.3	性能分析	35
4.4	主要故障与调试.....	35
4.5	实验进度	37

华中科技大学课程设计报告

5 设计总结与心得	39
5.1 课设总结	39
5.2 课设心得	39
参考文献.....	41

1 课程设计概述

1.1 课设目的

计算机组成原理是计算机专业的核心基础课。该课程力图以“培养学生现代计算机系统设计能力”为目标，贯彻“强调软/硬件关联与协同、以 CPU 设计为核心/层次化系统设计的组织思路，有效地增强对学生的计算机系统设计及实现能力的培养”。课程设计是完成该课程并进行了多个单元实验后，综合利用所学的理论知识，并结合在单元实验中所积累的计算机部件设计和调试方法，设计出一台具有一定规模的指令系统的简单计算机系统。所设计的系统能在 LOGISIM 仿真平台和 FPGA 实验平台上正确运行，通过检查程序结果的正确性来判断所设计计算机系统正确性。

课程设计属于设计型实验，不仅锻炼学生简单计算机系统的设计能力，而且通过进行中央处理器底层电路的实现、故障分析与定位、系统调试等环节的综合锻炼，进一步提高学生分析和解决问题的能力。

1.2 设计任务

本课程设计的总体目标是利用 FPGA 以及相关外围器件，设计五段流水 CPU，要求所设计的流水 CPU 系统能支持自动和单步运行方式，能正确地执行存放在主存中的程序的功能，对主要的数据流和控制流通过 LED、数码管等适时的进行显示，方便监控和调试。尽可能利用 EDA 软件或仿真软件对模型机系统中各部件进行仿真分析和功能验证。在学有余力的前提下，可进一步扩展相关功能。

1.3 设计要求

- (1) 根据课程设计指导书的要求，制定出设计方案；
- (2) 分析指令系统格式，指令系统功能。
- (3) 根据指令系统构建基本功能部件，主要数据通路。
- (4) 根据功能部件及数据通路连接，分析所需要的控制信号以及这些控制信号的有效形式；
- (5) 设计出实现指令功能的硬布线控制器；

华中科技大学课程设计报告

- (6) 调试、数据分析、验收检查;
- (7) 课程设计报告和总结。

1.4 技术指标

- (8) 支持表 1.1 前 27 条基本 32 位 MIPS 指令;
- (9) 支持教师指定的 4 条扩展指令;
- (10) 支持多级嵌套中断, 利用中断触发扩展指令集测试程序;
- (11) 支持 5 段流水机制, 可处理数据冒险, 结构冒险, 分支冒险;
- (12) 能运行由自己所设计的指令系统构成的一段测试程序, 测试程序应能涵盖所有指令, 程序执行功能正确。
- (13) 能运行教师提供的标准测试程序, 并自动统计执行周期数
- (14) 能自动统计各类分支指令数目, 如不同种类指令的条数、冒险冲突次数、插入气泡数目、load-use 冲突次数、动态分支预测流水线能自动统计预测成功与失败次数。

表 1.1 指令集

#	指令助记符	简单功能描述	备注
1	ADD	加法	指令格式参考 MIPS32 指令集, 最终功能以 MARS 模拟器为准。
2	ADDI	立即数加	
3	ADDIU	无符号立即数加	
4	ADDU	无符号数加	
5	AND	与	
6	ANDI	立即数与	
7	SLL	逻辑左移	
8	SRA	算数右移	
9	SRL	逻辑右移	
10	SU b	减	
11	OR	或	
12	ORI	立即数或	
13	NOR	或非	

华中科技大学课程设计报告

#	指令助记符	简单功能描述	备注
14	LW	加载字	
15	SW	存字	
16	BEQ	相等跳转	
17	BNE	不相等跳转	
18	SLT	小于置数	
19	STI	小于立即数置数	
20	SLTU	小于无符号数置数	
21	J	无条件转移	
22	JAL	转移并链接	
23	JR	转移到指定寄存器	
24	SYSCALL	系统调用	If \$v0==10 halt(停机指令) else 数码管显示\$a0 值
25	MFC0	访问 CP0	中断相关，可简化，选做
26	MTC0	访问 CP0	中断相关，可简化，选做
27	ERET	中断返回	异常返回，选做
28	lh	加载半字	
29	xori	异或立即数	
30	stliu	小于无符号数置 1	
31	bltz	小于 0 跳转	

2 总体方案设计

2.1 单周期 CPU 设计

单周期 CPU 采用分工合作的方式完成根据功能部件分为不同模块，每个人单独实现模块之后再综合。我们采用的单周期模块是我们小组汤汇川同学的单周期 CPU。NPC 模块的 verilog 代码，以及最后数据通路的搭建与基本测试也是由我来完成的。

总体结构图如图 2.1 所示。

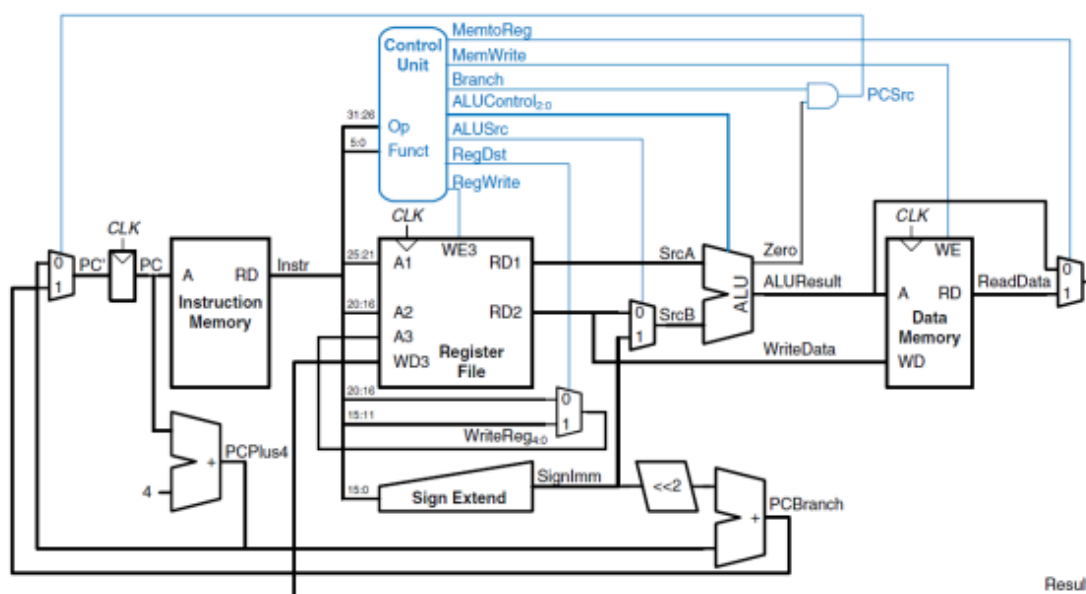


图 2.1 总体结构图

2.1.1 主要功能部件

我所实现的 PC 和 NPC 功能部件主要设计思路如下所示

1. 程序计数器 PC

程序计数器实际上就是一个 32 位寄存器，每当时钟周期上升沿到来的时候锁存输入的值。主要接受时钟脉冲，使能电平，下一个 pc 地址的输入，输出当前赔偿地址。

华中科技大学课程设计报告

2. 指令存储器 NPC

NPC 模块接受来自控制器的控制信号、当前 PC 和来自运算器的输出，输出 PC 的下一地址。本实验当中 NPC 需要考虑处理的跳转情况有两种 1) 直接跳转 (J/Jr/Jal 指令) 2) 条件分支 (bne/beq/bltz)。前者需要根据不同指令选择跳转地址，后者需要根据运算器计算跳转地址，比较是否需要跳转。其中对于 jal 指令还需要将当前 pc 送寄存器保存

2.1.2 数据通路的设计

如图下表 2.1 显示了单周期 cpu 的数据通路的组织情况。各个模块构建完毕之后，根据词表着手开始数据通路的构建。不需要关注具体指令的对应数据通路情况，只需要确定有哪几类输入输出即可。

表 2.1 数据通路表

指令	PC	IM	RF				Ext	ALU		DM	
			R1#	R2#	W#	Din		A	B	Addr	Din
Add	PC+1	pc	rs	rt	rd	ALU		RF-R1	RF-R2		
Add Immediate	PC+1	pc	rs		rt	ALU	s-imm	RF-R1	Ext		
Add Immediate Unsigned	pc+1	pc	rs		rt	ALU		RF-R1	Ext		
	pc+1	pc	rs	rt	rd	ALU		RF-R1	RF-R2		
And	pc+1	pc	rs	rt	rd	ALU		RF-R1	RF-R2		
And Immediate	pc+1	pc	rs		rt	ALU	z-imm	RF-R1	Ext		
Shift Left Logical	PC+1	pc	rt		rd	ALU	z-imm	RF-R1	Ext		
Shift Right Arithmetic	PC+1	pc	rt		rd	ALU	z-imm	RF-R1	Ext		
Shift Right Logical	PC+1	pc	rt		rd	ALU	z-imm	RF-R1	Ext		
Sub	pc+1	pc	rs	rt	rd	ALU		RF-R1	RF-R2		
Or	pc+1	pc	rs	rt	rt	ALU		RF-R1	RF-R2		
Or Immediate	pc+1	pc	rs		rt	ALU	z-imm	RF-R1	Ext		
Nor	pc+1	pc	rs	rt	rd	ALU		RF-R1	RF-R2		
Load Word	pc+1	pc	rs		rt	DM.Dout	s-imm	RF-R1	Ext	ALU	
Store Word	PC+1	pc	rs	rt			s-imm	RF-R1	Ext	ALU	RF-R2
Branch on Equal	pc+1/pc+1+OFFSET	pc	rs	rt				RF-R1	RF-R2		
Branch on Not Equal	pc+1/pc+1+OFFSET	pc	rs	rt				RF-R1	RF-R2		
Set Less Than	PC+1	pc	rs	rt	rd	ALU		RF-R1	RF-R2		
Set Less Than Immediate	PC+1	pc	rs		rt	ALU	s-imm	RF-R1	Ext		
Set Less Than Unsigned	PC+1	pc	rs	rt	rd	ALU		RF-R1	RF-R2		
Jump	PC&0xF000 address	pc									
Jump and Link											
Jump Register	rs	pc									
syscall (display or exit)			2	4							
Subu	PC+1	pc	rs	rt	rd	ALU		RF-R1	RF-R2		
Xori	PC+1	pc	rs		rt	ALU	z-imm	RF-R1	Ext		
Load Half Unsigned Word	PC+1	pc	rs		rt	DM.Dout	s-imm	RF-R1	Ext	ALU	
	PC+1/PC+SignExtImm	pc	rs	rt				RF-R1	RF-R2		
	4个	pc	3个	2个	2个	2个	2个	RF-R1	2个	ALU	

2.2 中断机制设计

2.2.1 总体设计

在单周期 CPU 上完成多级中断的设计，能在单周期 CPU 中设计多级嵌套中断处

华中科技大学课程设计报告

理机制，能处理多个外部中断事件，能正确的终止主程序的执行，转为为按钮事件服务的中断服务程序，中断服务程序执行完毕后应返回主程序继续运行，不同的按钮会进入不同的中断服务程序，高优先级中断可以打断低优先级中断，高优先级中断服务程序执行完毕后应能返回被中断的中断服务程序，直至主程序。

中断过程主要可以分位为如下一些步骤：1) 中断捕获 2) 中断仲裁 3) 中断识别 4) 中断响应 5) 中断服务，多级中断的具体流程如图 2.2 所示。

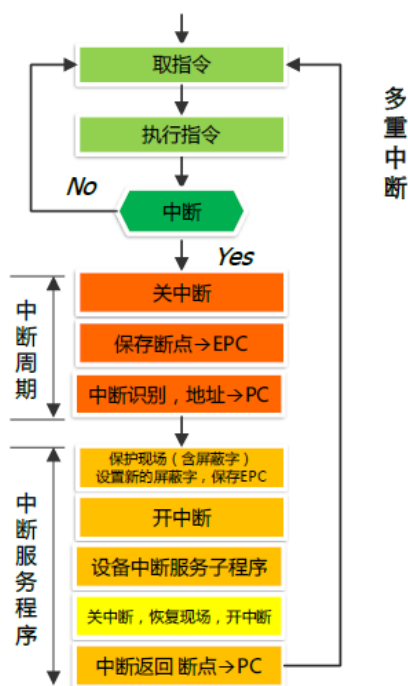


图 2.2 多级中断流程图

单周期中，总是在一条指令结束之后检查是否有中断出现。若没有继续下一条指令的执行，若有则进入中断周期。在终端周期当中需要首先关中断防止保护现场的功能被新的中断打断。然后保存断点，进行中断识别跳转到响应的中断服务子程序。中断服务子程序先使用堆栈保护寄存器现场，再开中断，允许自身被打断，运行中断服务程序。结束时关中断之后再恢复现场，最后中断返回。

中断的整个流程需要软硬件配合，中断周期由硬件自动完成，中断服务程序当中，软件指令负责保护寄存器现场，并需要通过软件指令和硬件交互完成开中断和关中断。中断返回指令由中断服务程序发出，由硬件实现断点送 PC，同时相应中断寄存器清零等操作。

2.2.2 硬件设计

需要保存的状态有中断请求寄存器 **IR**：标识那些中断源有请求、中断使能寄存器 **IE**：表示当前是否允许中断打断，异常程序计数器 **EPC**：储存断点处的 **PC** 值。考虑到一个问题：如何防止自己打断自己，增设一个寄存器 **curIR**：表示当前执行的中断的编号。同时为了实现多级中断同时简化实现，设置一个硬件堆栈来在嵌套中断到来的时候总是能保存当前中断的编号。

1. 中断捕获模块

中断的按钮信号是异步的，因此一方面我们需要将其同步化，另外一方面，中断信号在得到响应之前不应该消失，因而应该设置中断请求寄存器 **IR** 来保存中断的信息。对应中断请求寄存器的值在中断返回时被清除。

2. 中断仲裁与中断识别

由于我们只要实现对三个中断源的优先判断：则通过一个优先编码器则可以选择当前优先级最高的中断。为了简化中断识别的实现，使用一个多路选择器，用中断号选择程序入口地址。

3. 中断响应

中断响应过程中包含中断仲裁和中断识别，这两者都可以用组合电路的逻辑试下。同时在中断响应的中断周期当中需要保存当前 **PC** 到 **EPC**，关中断，设置 **IE** 为 1，中断服务程序入口送到 **PC** 当中，这些均可以在一个周期当中完成。

4. 开关中断、中断返回与指令支持

涉及到硬件和软件的协同合作，硬件需要能够接受软件的命令。具体需要支持的操作有 1) 根据传入的值设置 **IE** 的值 (etc) 2) 根据传入的值恢复 **epc** 的值(etc) 3) 中断返回 **epc** 值送 **pc** 值

2.2.3 软件设计

为了实现软件控制开关中断与现场保护，增加一组新的指令，以对 `cp0` 寄存器进行操作。指令集的格式与其含义如表 2.2 所示。

表 2.2 中断指令表

#	指令格式	简单功能描述	备注
1	<code>mfc0 \$x1, \$x2</code>	访问 <code>cp0</code> 寄存器组 <code>x2</code> 号寄存器并将其值保存在通用寄存器 <code>x1</code> 中	约定 <code>x2</code> 为 0 时代表 IE，为 1 时代表 EPC
2	<code>mtc0 \$x1, \$x2</code>	送通用寄存器 <code>x1</code> 的值到 <code>cp0</code> 寄存器 <code>x2</code> 当中	同上
3	<code>eret</code>	中断返回	

中断程序中，使用了什么寄存器就在保护现场阶段保护什么寄存器。

保护现场需要使用堆栈，堆栈采用向下增长的模式，因此需要在程序一开始给 `sp` 设置一个较高的值，以一个较高的地址作为堆栈的基地址，为堆栈留下足够的空间。

2.3 流水 CPU 设计

2.3.1 总体设计

流水线为了突破单周期关键路径较长所造成的频率限制，将流水线技术应用于指令的解释执行过程，形成了指令流水线。

MIPS 指令流水线通常将指令执行过程细分为取指令 IF、指令译码 ID、指令执行 EX、访存 MEM、写回 WB 共五个阶段，在每个阶段的后面都需要增加一个锁存器（又称为流水接口部件，用于锁存本段的处理完成的数据或结果），以保证该阶段的执行结果给下一个阶段使用，如图 2.3 所示。

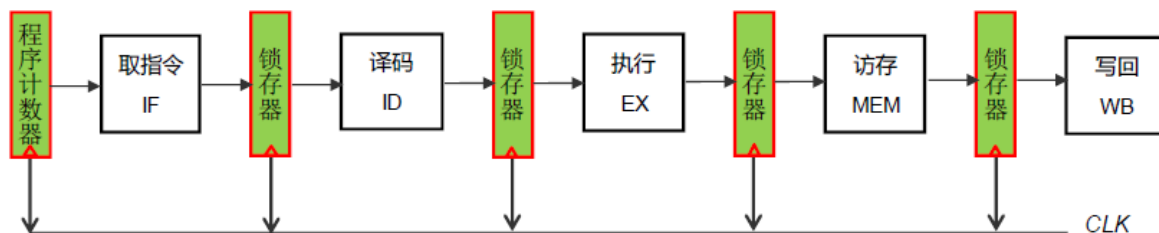


图 2.3 指令流水线逻辑结构

华中科技大学课程设计报告

如图 2.3 所示，程序计数器、所有锁存器均采用公共时钟同步，每来一个时钟，各段逻辑功能部件处理完成的数据会锁存到锁存器中，作为下段的输入，指令的执行进入下一段。由于五个阶段逻辑延迟时间并不一致，为保证指令流水线正确运行，最大时钟频率取决于五个阶段中最慢的阶段，所以分段时应该尽量让各阶段时间延迟相等，假设各段时间延迟均为 T 。锁存器在公共时钟的驱动下可以锁存流水线前段加工完成的数据以及控制信号，锁存的数据和信号将用于后段的继续加工或处理。

2.3.2 流水接口部件设计

流水接口部件采用同步清零方式，流水线内部应该时一组寄存器来锁存前一个部分传给其的值。

各段流水所保存的值如表 2.3 所示。

表 2.3 各段流水接口输入表

流水接口输入名称	位宽	释义
IF/ID 段		
PC	32	当前 PC 的值
IR	32	当前指令的机器码
ID/EX 段		
PC	32	当前 PC 的值
Aluop	5	ALU 的操作码
Alusrc, rWE, jump, beq, bne, memw, jr, bltz, lh, memread	1	Alu 来源选择, 寄存器写使能, jump 指令标识, beq 指令标识, bne 指令标识, memw 内存写使能, jr 指令标识, bltz 指令标识, lh 指令标识, memread 读内存标识
rW	5	写寄存器编号
RA, RB	32	RA, RB 寄存器值
pcjump	32	Jump 指令跳转地址
imm	32	I 型指令立即数扩展
Hault	1	停机信号
EX/MEM 段		
result	32	ALU 运算结果

华中科技大学课程设计报告

RB	32	同 ID/EX 段
memW, rWE, hault,lh,memread	1	同 ID/EX 段
rW	5	同 ID/EX 段
MEM/WB 段		
memvalue	32	读内存段的值
result	32	同 EX/MEM 段
memread, rWE, hault	1	同 EX/MEM 段
rW	5	同 EX/MEM 段

2.3.3 理想流水线设计

IF 段包含 ROM, PC, 向下一段输出当前 pc 的值, 当前指令的机器码。

ID 段包含 REG, CONTORL, DECODER, Syscall (处理中断), 解析指令格式, 生成控制信号, 读取寄存器的值。向下一段输出控制信号、寄存器 RA、RB 的值, Syscall 输出的停机信号, decoder 解析出的立即数, ALU 操作码, 写寄存器的编号。

EX 段包含 ALU, NPC, ALU 根据操作码进行运算, NPC 根据输入 PC、立即数、控制信号判断跳转, 计算下一个 pc 地址。向下一段输出寄存器 B 的值, ALU 计算结果, 写 register 编号, 控制信号等。

MEM 段包含 ram, 根据控制信号读写存储单元。想下一段输出读取的内存单元值, 控制信号, ALU 运算结果, 写寄存器编号。

同时由于理想流水线当中不存在分支指令, npc 部分可以暂时不用实现。

流水线各段的输入为前一段流水部件的输出, 输出到后一段流水线输入, 具体见表 2.3。

2.4 气泡式流水线设计

2.4.1 总体设计

理想流水线所有待加工对象均需要通过相同的部件(阶段), 不同阶段之间无共享资源, 且各段传输延迟一致, 进入流水线的对象也不应受其他功能段的影响, 但这仅

华中科技大学课程设计报告

仅适合工业生产流水线，计算机指令流水线存在较多的指令相关，会引起流水线的冲突和停顿。

指令相关，是指指令流水线中，如果某指令的某个阶段必须等到它前面的某条指令的某个阶段完成才能开始，也即是两条指令间存在着某种依赖关系，则两条指令存在指令相关。指令相关包括数据相关、结构相关、控制相关，指令相关会导致流水线冲突/冒险（Hazzard）。

为了解决相关所带来的不一致性问题，引入气泡，即流水线暂停等待前一条结果执行完毕得到相应的值。

总结如下 2.4.2 – 2.4.4 的方案，主要在 IF/ID 和 ID/EX 流水部件处增加气泡，具体增加气泡的逻辑如下表 2.4 所示

表 2.4 插入气泡模块模块输入输出

输入名称	位宽	说明
rA, rB	32	ID 段选择寄存器 A 和寄存器 B 的编号
rweEX, rweMEM	1	EX 段和 MEM 段指令的寄存器写使能
rWEX, rWMEM	5	EX 段和 MEM 段指令的寄存器写编号
rBValid	1	ID 段 rB 是否合理
JMP	1	由 JMP, bSuc 相或作为输入，指示是否成功跳转
输出名称	位宽	说明
BubId, BubIf	1	控制 IF/ID 和 ID/EX 流水部件是否同步清零
stall	1	输出当前是否为数据通过，以正确计数

2.4.2 解决数据相关

数据相关即为，当前指令要用到前面指令的操作结果，而这个结果尚未产生或尚未送达指定的位置，会导致当前指令无法继续执行。可以通过插入气泡直到后面的指令全部执行完即可。可以在设置检测逻辑，总是检测 ID 段和 EX, MEM 段是否有数据相关，若存在数据相关，下一周期则插入气泡。

2.4.3 解决结构相关

结构相关在于，在同一周期要对同一个部件进行操作的冲突。单周期流水线代码段和数据段分离的结构本身就解决了存储器的结构冲突，另外一个结构冲突在于可能会同时读写寄存器文件，通过规定写总是在时钟周期下降沿，读在上跳沿解决，通过半周期的错开来解决结构冲突。

2.4.4 解决控制相关

由于遇到跳转指令时，是否能成功跳转到 EX 段才能算出来（这也是为什么 NPC 模块需要放在 EX 段），若跳转到 EX 段时前面已经误取了两条错误的指令，不能允许这两条指令执行下去。此时就要把这两条指令清为气泡，并将 pc 置为正确的值。于跳转分支之前的指令可以安全地让其执行下去。

2.5 数据转发流水线设计

2.5.1 总体设计

气泡流水线通过延缓 ID 段取操作数动作的方式解决数据冲突问题，但大量气泡的插入会严重影响流水线的性能，还有一种思路是先不考虑 ID 段所取的操作数是否正确，而是等到 EX 段实际需要使用这些操作数时再考虑正确性问题。通过这种思路，我们只需要等到 EX 段利用从 RA 和 RB 取出的值时再确定其正确性，不正确就使用正确的值进行替换。因而主要的替换对象是 EX 段 RA 和 RB 的值。

2.5.2 EX 段重定向设计

为了替换正确替换 RA 和 RB 的在 EX 段的值，需要向 EX 段传入更多的信息，具体如下表所示。同时，由于 SYSCALL 也需要用到从寄存器当中读取出来的值，SYSCALL 利用的值也需要被重定向，为了方便起见移动 SYSCALL 模块到 EX 段。

2.5.3 气泡模块重新设计

由于重定向解决了大部分数据相关问题，只有在 `loaduse` 情况下才需要插入一个气泡。因而修改气泡插入模块只需要对 ID 和 EX 段的数据冲突检测是否为 `loaduse` 即可。有关控制冲突的部分可以不用变。

2.5.4 重定向模块设计

重定向模块接受来自 EX、MEM、WB 的输入，输出到 EX 段中重定向后的 RA 和 RB，接口定义如表 2.5 所示。

表 2.5 重定向模块输入输出

输入名称	位宽	说明
rAEX, rBEX	32	EX 段指令选择寄存器 A 和寄存器 B 的编号
rBValidEX	1	EX 段当中的指令是否用到 rB
rweWB, rweMEM	1	WB 段和 MEM 段指令的寄存器写使能
rWWB rWMEM	5	WB 段和 MEM 段指令的寄存器写编号
RAEX, RBEX	32	EX 段 RA 寄存器和 RB 寄存器的值
ResultMem, ResultWB	32	Mem 段的 alu 运算结果，WB 段的访存选择之后的结果
输出名称	位宽	说明
X, Y	32	分别对应于重定向之后的 RA 和 RB

3 详细设计与实现

3.1 单周期 CPU 实现

3.1.1 主要功能部件实现

1) 程序计数器 (PC)

① Logism 实现:

使用一个 32 位寄存器实现程序计数器 PC，触发方式为下降沿触发，输入为下一条将要执行的指令的地址，输出为当前执行指令的地址。Halt 为停机信号，将此控制信号通过非门取反之后和时钟相与，当需要进行停机时，Halt 控制信号为 1，经过非门之后为 0，与时钟信号相与，屏蔽时钟信号，使整个电路停机。如图 3.1 所示。

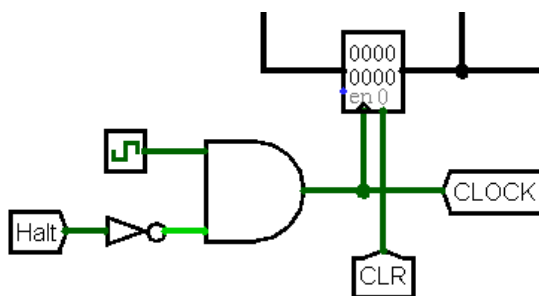


图 3.1 程序计数器 (PC)

② FPGA 实现:

程序计数器 PC 的 Verilog 代码如下:

```
module PC ( nextpc_in, enable_in, clk_in, rst_in, pc_out);  
    input wire [31:0] nextpc_in;  
    input wire enable_in;  
    input wire clk_in;  
    input wire rst_in;  
    output reg [31:0] pc_out;  
  
    initial begin  
        pc_out = 0;  
    end  
end
```

```
always @(posedge clk_in) begin
    if ( rst_in )
        pc_out = 32'h00000000;
    else if ( enable_in )
        pc_out = nextpc_in;
    else
        pc_out = pc_out;
end

endmodule
```

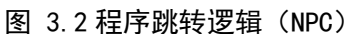
这里程序计数器 PC 相当于是被封装成了一个 32 位带使能控制的寄存器。

2) 程序跳转逻辑 (NPC)

① Logism 实现:

程序跳转逻辑 NPC 被单独封装成了一个模块。pc 是当前 PC 的值, pcjump 是 j 指令跳转的地址值, jump、equal、beq、bne_in、bgez_in、jr 分别是指示对应指令的信号, extendimm 是分支指令的偏移地址, da 是 jr 指令跳转的目标寄存器的值, pcjar 是 jal 指令时送入 31 号寄存器存储的寄存器的值, nextpc 是 NPC 决定的下一个 PC 的地址。

如图 3.2 所示。



华中科技大学课程设计报告

```
assign jump_addr[31:28] = pcjar_out[31:28];
assign jump_addr[27:0] = pcjump_in;
assign branch_addr = (extendimm_in << 2) + pcjar_out;

assign branch_control = (bgez_in & cmp) | (bne_in & (~equal_in)) | (beq_in &
equal_in);

// jr select
MUX2_32 jr_select (.in0(jump_after_addr), .in1(da_in),
                  .control(jr_in), .out(nextpc_out));

// jump select
MUX2_32 jump_select (.in0(branch_after_addr), .in1(jump_addr),
                   .control(jump_in), .out(jump_after_addr));

// branch select
MUX2_32 branch_select (.in0(pcjar_out), .in1(branch_addr),
                     .control(branch_control), .out(branch_after_addr));

endmodule
```

3.1.2 数据通路的实现

本次课程设计采用的工程化的设计模式，一次性构建所有的数据通路。主要实现方法为，对于每一条指令，将其改写成 RTL（Register Transfer Level），忽略控制类信号，仅保留数据类信号，根据 RTL 功能填写对应指令的数据通路表，描述五大部件之间的连接关系，记录各部件输入端数据来源。

根据总体方案设计中数据通路设计那一小节的详细内容，具体分析每一条指令在执行过程中各个主要部件的输入和输出端口的连接，完成指令系统数据通路表的填写，如表 2.1 所示。

① Logism 实现：

在完成指令系统数据通路表的填写之后，根据列出的数据通路表，进行多指令数

响应周期时, hasIR 为 1, 将中断程序入口地址送入 next_pc, 将当前 pc 地址写入 EPC, 当中断返回时, ERET 为 1, 将保存的断点地址 EPC 送回 next_pc。

4. 中断仲裁、中断识别、与中断现场保护

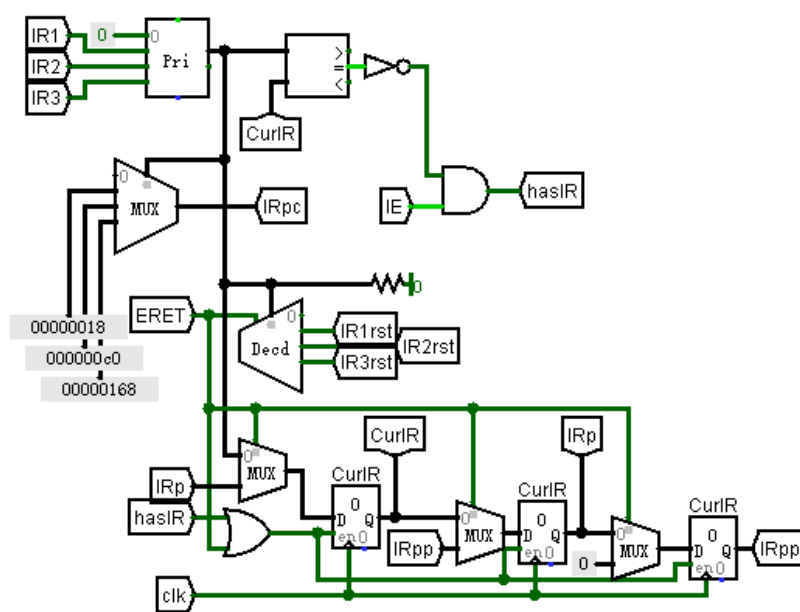


图 3.8 中断仲裁 & 中断识别

如图 3.8 所示, 得到中断请求寄存器之后使用优先编码器获取最高优先级中断, 并于当前中断 (无中断当前中断号为 0) 比较, 若不同而且中断使能寄存器为 0, 则表示当前进入中断响应的周期, hasIR 为 1。根据终端号选择中断程序入口地址输出到 IRpc。最下面 3 个寄存器构成一个大小为 3 的硬件堆栈, 每当中断响应周期时, 其将当前中断号压入堆栈, 当中断返回 eret 时则反向出栈, 那么栈顶即最左边寄存器时钟保存当前中断号。同时, eret 时根据终端号清零对应中断请求寄存器。

3.2.2 中断程序的软件实现

中断程序的汇编代码如下面示例代码所描述:

```
IR1:
    # 保护现场
    addi$х, $sp, -4
    sw $t1, 0($sp)
```



```
.....  
# 保存 EPC  
mfc0    $t0, $1  
addi     $sp, $sp, -4  
sw  $t0, 0($sp)  
# 开中断  
addi     $t0, $zero, 0  
mtc0     $t0, $0  
  
# 中断服务程序  
.....  
  
# 恢复 epc  
lw  $t0, 0($sp)  
addi$sp, $sp, 4  
mtc0     $t0, $1      #restore epc  
# 恢复现场  
lw  $x, 0($sp)  
addi$sp, $sp, 4  
.....  
# 中断返回  
eret
```

上述代码实际上重现了图 2.2 中中断子程序的流程，一些细微的差别在于最后关中断恢复现场之后再开中断的工作留给 `eret` 一并实现了。有关 `mfc0`、`mtc0`、`eret` 的含义参见表 2.2。

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

一个流水同步清零部件接口的例子如图 3.9 所示

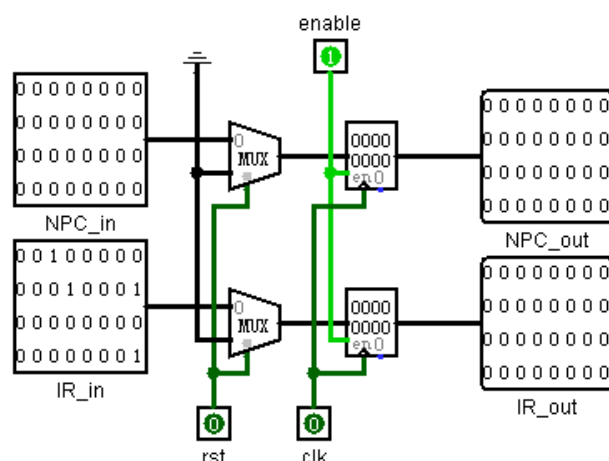


图 3.9 IF/ID 流水部件

图 3.9 中，当 rst 为 1 时，下一个时钟周期上升沿脉冲到来时流水部件中寄存器的值被清零，否则流水总是正常的写入输入的值，其它流水结构除了寄存器位数上面可有差异以外与该结构完全相同，具体位数与含义见表 2.3 各段流水接口输入表。

3.3.2 理想流水线实现

理想流水线实现如下图所示：

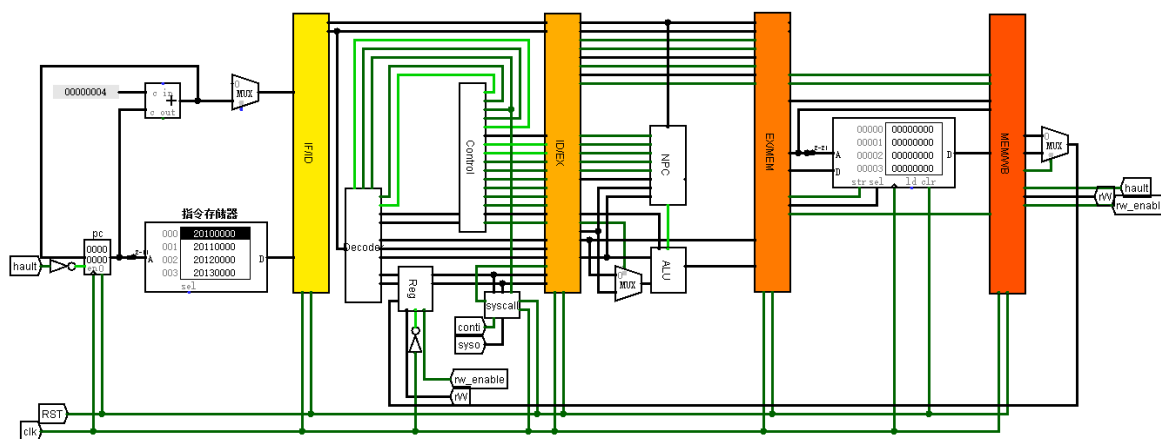
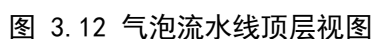
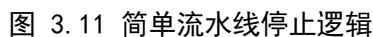


图 3.10 理想流水线

其中，由于不存在分支指令，将 pc 每次简单地设置成了 $pc+4$ 。

数据通路构建逻辑大致与单周期相同，总是将每一段处理指令的信号通过流水线单独传输，因而实现五段流水实际上都在处理着不同的指令。

流水线地停止逻辑如图 3.11 所示。当在 ID 段处理的停机信号随着流水线传到 WB 段后，下一个时钟周期流水线被排空同时锁存停机信号。PC 使能段变为 0，不再变化从而实现停机。



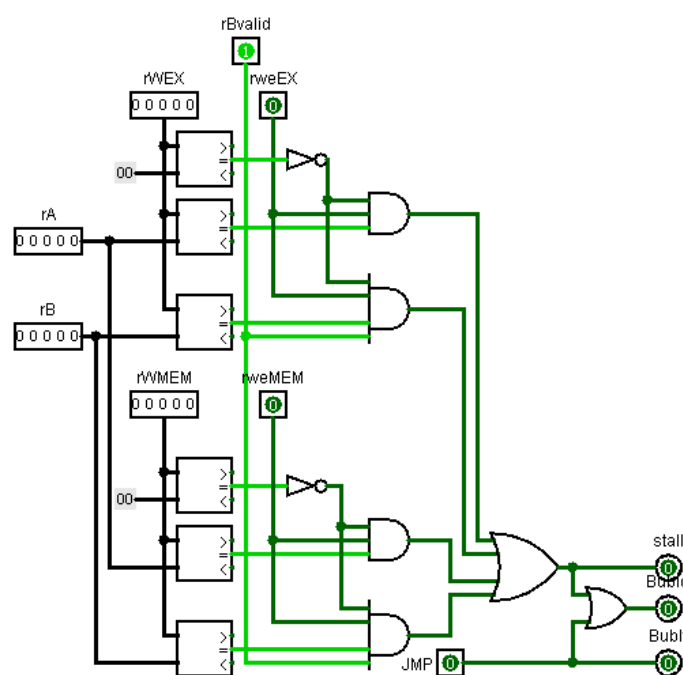


图 3.13 气泡插入模块

各个输入输出的意义见表 2.4。

当 stall 为 1 的时候表明有数据冲突，需要检测 MEM 段和 EX 段的写目标寄存器 rW 和 ID 段的读寄存器 rA、rB 之间是否存在冲突。因而需要检测四组冲突存在。同时，当 MEM 段和 EX 段指令不发生写寄存器操作时不会产生冲突，检测与 rB 冲突，当 ID 段指令不存在读第二个寄存器 rB 时不会产生冲突。由于零号寄存器不会产生冲突，所有对零号寄存器的读写冲突均忽略。

当发生控制冲突的时候，BubId 和 BubIf 均为 1，F/ID 和 ID/EX 流水中保存的指令将被清零。

气泡产生模块在流水线中的结构如图 3.14 所示。

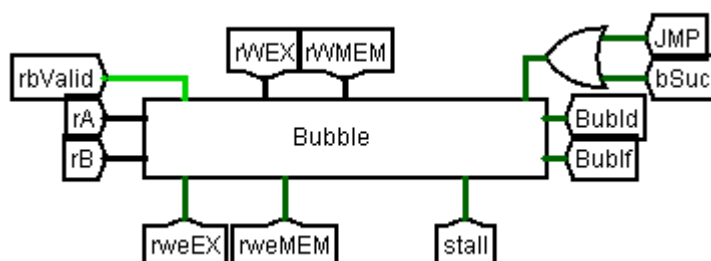


图 3.14 气泡插入部分顶层结构

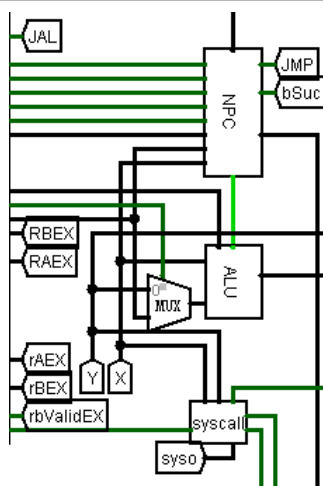


图 3.17 重定向 EX 段

如图 3.17,X 和 Y 分别对应于重定向后的 RA 和 RB。

3.5.2 气泡模块的重新设计。

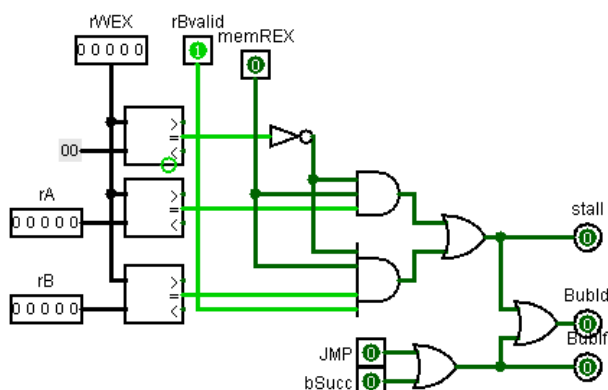


图 3.18 重定向流水线 load-use 气泡

如图 3.18 所示为重定向流水线中的气泡模块，其通过检测 memREX 和是否存在 ID 和 EX 段的数据相关来检测是否有 loaduse，若存在 loaduse，则插入一个气泡。

3.5.3 重定向模块

其顶层结构如图 3.19 所示，相关接口定义如表 2.5 所示。

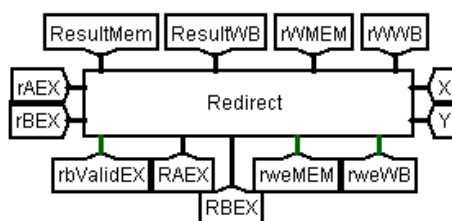


图 3.19 重定向模块顶层结构

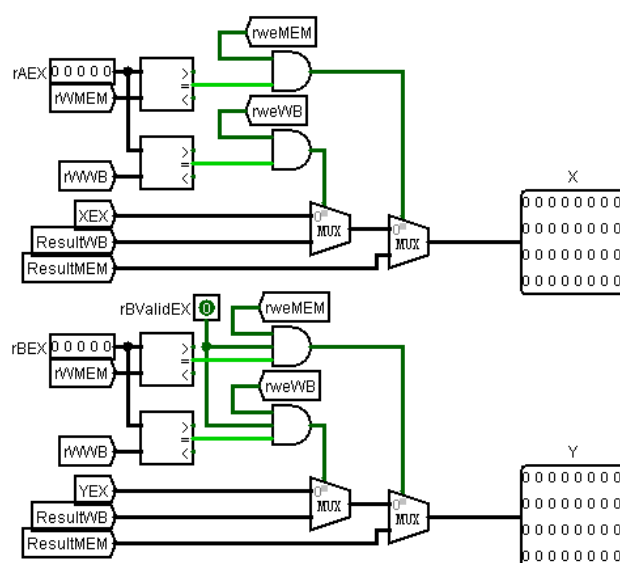


图 3.20 重定向模块实现

如图 3.20 是重定向模块的实现，将 rWmem 和 rWWB 依次与 EX 段的 rArB 比较得到是否有写后读的情况。若有进行重定向。注意一个细节，来自 MEM 段的数据重定向优先级比 WB 段要高。这是由于 WB 段指令在前，MEM 段在后。

3.6 重定向流水线上板实现

代码层次结构如图 3.21 所示。

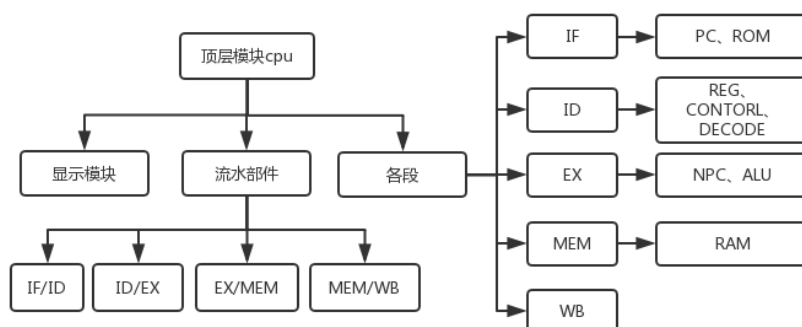


图 3.21 代码模块层次结构图

顶层模块为 cpu，利用流水部件模块，IF、ID、EX、MEM、WB 等模块构建顶层数据通路。各段内再利用各功能模块构建底层数据通路。

详细代码见代码清单。

4 实验过程与调试

4.1 测试用例和功能测试

已知单周期 cpu benchmark 运行总周期为 1544。

NPC 在 EX 段，指令误取深度为 2。

4.1.1 测试用例 1 benchmark

1. 气泡流水线

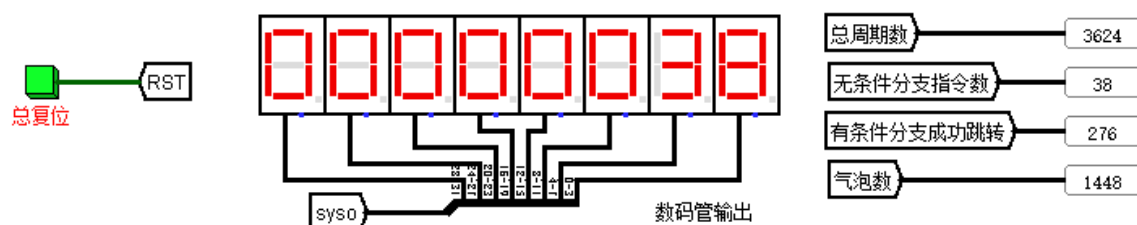


图 4.1 benchmark 气泡流水线

气泡流水线的 benchmark 测试程序如图 4.1 所示。

气泡流水线周期数=单周期执行周期数 + (流水冲满时间 - 1) + J 指令*误取深度 + 条件分支成功次数*误取深度 + 气泡数

$$3624 = 1544 + (5-1) + 38*2 + 276*2 + 1448$$

实验检查通过，实验结果符合周期公式预期。

2. 重定向流水线

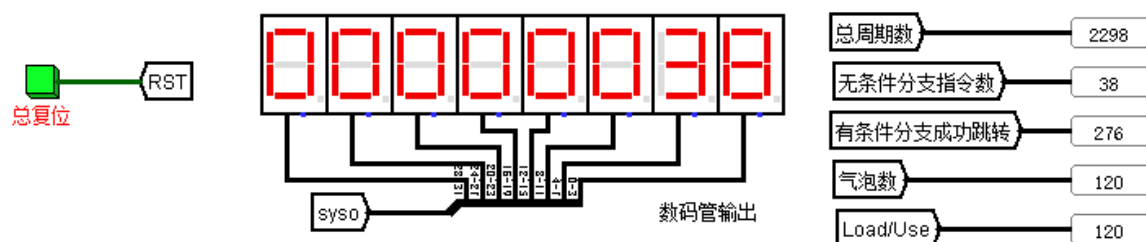


图 4.2 benchmark 重定向流水线

重定向流水线的 benchmark 测试结果如图图 4.2 所示。

华中科技大学课程设计报告

重定向周期数=指令条数 + 流水冲满时间 + 分支冲突次数*n+LOAD-USE 数

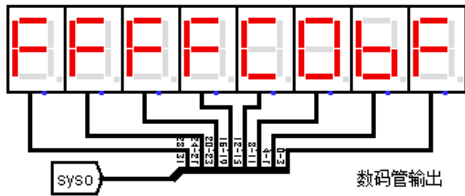
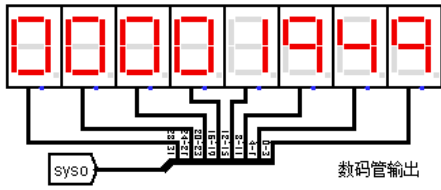
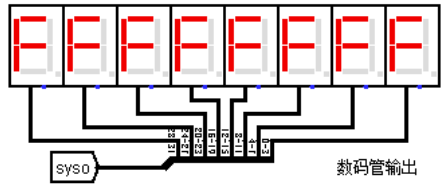
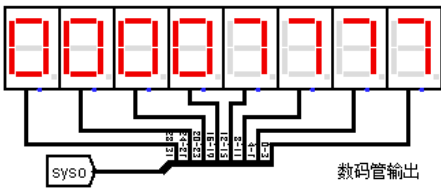
$2298 = 1546 + 4 + (276+38) * 2 + 120$

实验检查通过，实验结果符合周期数公式预期。

4.1.2 测试用例 2 ccmb

ccmb 在重定向流水线上的测试结果如表 4.1 所示

表 4.1 ccmb 测试结果

指令	测试结果	描述
lh		从 FFFF8281 开始到 FFFF8483..... 一直增加到 FFFFC06F
xori		从 1997 开始递减到 1949
stliu		从 FFFFFFFF1 开始递增到 FFFFFFFF
bltz		数字 00007777 和数字 00008888 反 复交替变化

4.2 课设思考题与主要问题的记录

4.2.1 Mips 单周期 CPU 上板

经验记录

- (1) 不能为了编程的方便使用时序逻辑模拟组合逻辑，就算通过仿真仍然会让模块留下时序逻辑的 bug。经验是完全仿照 logisim 的原理图，只有在寄存器的部分使用 reg 类型变量，其它部分全部使用 wire 类型。
- (2) 为了保证模块之间参数传递的正确性使用 .parameter_name(p1) 的方式连接模块接口。
- (3) verilog 在位宽不匹配的时候只会自动舍去或者扩展，对于综合生成的警告需要仔细阅读。

4.2.2 理想流水线

- (1) 为何要把地址转移逻辑放在 EX 段？

地址转移逻辑可以放在 ID 段也可以放在 EX 段，但是由于地址转移逻辑可能需要用到 RB 的值，若放在 ID 段，之后实现重定向的时候需要单独为地址转移逻辑实现一套重定向机制。

- (2) 关于停机信号的产生与识别

syscall 模块安排在了 ID 段，ID 段产生停机信号之后停机信号需要随着流水线传输下去直到流水线被排空，这个时候停机指令与前面的所有指令均执行完，可以停机。

- (3) 关于继续执行和停机功能

实现继续执行功能只需要设置一个按钮绑定到控制流水线停止的寄存器的清零端，即可通过异步清零停止对 pc 的 disable。当生成停机信号的时候 pc 是否应该立即停止加载新的指令呢？答案是不必，由于停机信号后面的指令是继续执行时需要执行的指令，停机信号造成流水线的停止只是将这些值锁存在流水线的流水部件当中，当恢复执行的时候，这些指令恢复流动。

4.2.3 气泡流水线

实验思考题

- (1) 插入气泡能否直接使用寄存器的异步清零信号，为什么？

不能，首先使用异步清零方式可能会将本来应该流向下一个流水部件的指令清零，就算增加了一个寄存器强行同步化异步清零，在实现解决控制冲突的时候也会造成周期数不一致的情况。正确方法是使用同步清零，下一个时钟周期脉冲到来的时候同时插入气泡，并将当前指令传给下个流水部件。

华中科技大学课程设计报告

(2) 插入气泡的个数是否需要用电路进行控制？为什么？

不用，总是使用组合逻辑判断是否存在数据相关，存在时，总是在下一个周期插入一个气泡。控制相关跳转成功时同时向 IF 和 ID 段插入气泡即可（指令清空）。

(3) MIPS 零号寄存器是比较特殊的寄存器，如果源操作数是零号寄存器时是否存在数据相关？你是如何解决的？气泡指令之间是否存在相关？

由于零号寄存器值不会被改变，直接忽视掉零号寄存器造成的数据相关即可。这也解释了为什么气泡不会和任何指令之间产生数据相关，全 0 的指令含义是 `sll $0,$0`

(4) 采用气泡方式解决数据相关后测试标准测试程序 `benchmark` 时钟周期为什么反而比单周期 CPU 多了很多，难道流水线还不如单周期 CPU 吗

气泡流水线为了解决数据相关和控制相关使用了较简单的策略，流水线停滞导致了较大的性能损失，但是由于分段其关键路径时延仍比单周期流水线要小的多，因而比理想流水线能达到更大的频率。

经验记录

(1) 一开始使用非同步清零的气泡流水线也通过了测试，只是数据冲突需要加一个寄存器同步化清零，控制冲突时需要增加一个反馈回路生成一个清零脉冲。跟老师沟通之后得知这种设计可能引入潜在的问题，由于最小时钟周期不再是时钟信号跳变。

4.2.4 重定向流水线

实验思考题

(1) 重定向逻辑放在流水线哪个阶段更好，为什么？

一种思路是用时再重定向，则可以将所有重定向工作全部放在 EX 段，这样做的优点是简单方便。另一种思路是可以再 ID 段重定向，这样 NPC 也可以提前到 ID 段，那么误取深度就能降低，但是这样存在的问题在于重定向时 NPC 需要的 EX 的指令结果并没有算出来，需要增加运算部件。

4.2.5 多级中断与流水中断

(1) MIPS CPU 函数调用与中断服务程序的执行有多大区别，有否共同之处？

都涉及到 PC 值的改变与返回，函数调用断点保存在 31 号寄存器当中（`jal`），中断服务程序断点保存在 EPC 当中

华中科技大学课程设计报告

(2) 开中断，关中断如何实现？

首先要设置 1 位的 CP0 寄存器当中的使能寄存器 IE，然后涉及软件指令改变寄存器的状态。

(3) CPU 如何判断当前有中断需要响应？

我设置了一个保存当前中断号的寄存器，中断从 1 开始编号，无中断时中断寄存器内值为 1，中断响应寄存器有信号而且当前最高优先级中断号和当前中断编号寄存器值不同时表示有中断要响应（或者优先级高打断低）

(4) 最终实现的单级中断处理器中中断隐指令的开销是多少个时钟周期？

我的实现在一个时钟周期内同时完成了 PC->EPC，中断地址->pc，以及关中断。

(5) 多级嵌套中断的断点如何处理

所有断点都保存到 EPC 当中，允许自身被打断之前先要利用堆栈保存自身的断点

(6) 单周期 CPU 中断处理和流水中断处理有何区别？

单周期总是在指令执行之后检测中断，多周期同时有五条指令在执行，因而暂停执行某些指令，其它指令怎么处理需要考虑。

4.3 性能分析

不同方案的总时钟周期数等参数如表 4.2 所示。

表 4.2 benchmark 时钟周期统计

流水线类型	总时钟周期	控制冲突次数	气泡数 / load-use
单周期	1546	-	-
气泡流水线	3624	314	1448
重定向流水线	2298	314	120

可以看到重定向流水线比起气泡流水线节省了大量的气泡周期数，误取深度为 2 控制冲突仍然导致了大量的周期浪费，若采用动态分支预测总周期数可以到 2000 一下，进一步提高流水线的性能。

4.4 主要故障与调试

4.4.1 JaL 跳转问题

重定向流水线： JAL 指令无法正常跳转。

故障现象：执行 J 指令测试时，JAL 指令无法正确返回，程序陷入死循环。

原因分析：JAL 指令没有成功将 PC+4 的值写入 31 号寄存器

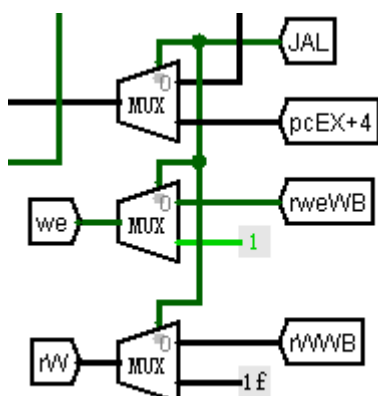


图 4.3 JAL 跳转故障问题

如图 4.3 之前为了解决 JAL 的跳转写入问题，直接在 EX 段将输出引出到 WB 段然后进行一个片选。然而这就引入了问题，JAL 指令会与同一时刻在 WB 段中的指令发生冲突，WB 段的指令不能正确写入。显然这种做法是错的图 4.3 所示的数据通路应该被删去。

解决方案：解决这个问题的方法在于让流水线处理 JAL 写入寄存器这一操作，那么就要根据 JAL 的控制信号决定是传 alu 计算结果还是传当前 PC+4。这一数据通路部分加在 EX 段，解决方案如图 4.4 所示。

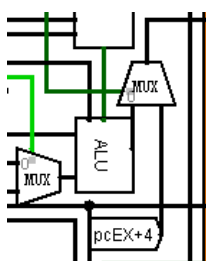


图 4.4 解决 JAL 跳转问题

4.4.2 插入气泡时丢失指令

气泡流水线中，气泡插入导致后续指令错误

故障现象：气泡流水线中，插入气泡后后续指令没能正确执行，查看后续指令 pc 地址发现后续指令跳过了一条。

原因分析：插入起跑时前方指令没有正确停止下来，可能忽略了响应流水部件和 pc 使能段的控制。

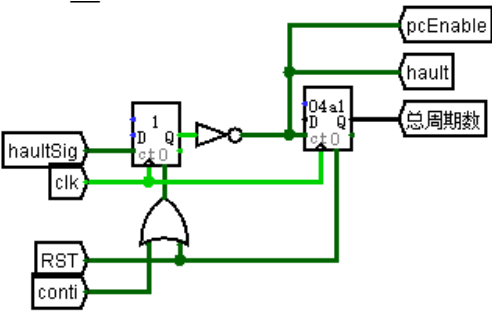


图 4.5 气泡流水线指令丢失错误

如图 4.5 所示，只有当停机时 pcenable 才为 0，pcenable 是设置错误。

解决方案：

如图 4.6 所示，加一个与门与上气泡信号的非。则当有气泡的时候输出同样为 0。

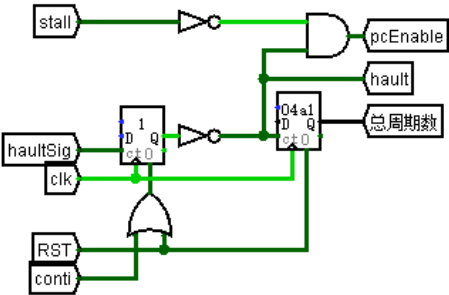


图 4.6 气泡流水线指令丢失解决方案

4.5 实验进度

表 4.3 课程设计进度表

时间	进度
第一天	小组讨论选出单周期 CPU 模板，对单周期模板进行修改和封装，分配单周期上板的任务。
第二天	完成单周期 CPU 自己负责的模块 NPC、PC，并完成构建数据通路，完成整合其它人的代码，开始调试，通过前仿真。
第三天	解决单周期上板 fpga 中的 bug，通过检查，开始理想流水线。
第四天	理想流水线调试通过，通过检查，开始气泡流水线的 Logisim 实现。
第五天	完成气泡流水线的实现，完成气泡流水线的调试，为气泡流水线增加 ccmb，气泡流水线通过检查，完成重定向流水线的实现，开始重定向流水线的调试
第六天	完成重定向流水线的调试，开始多级中断的学习和实现
第七天	完成多级中断的调试，通过测试，开始流水中断实现

华中科技大学课程设计报告

时间	进度
第八天	完成流水中断实现调试，通过检查，开始重定向流水线上板 verilog 代码的编写。
第九天	完成重定向流水线上板代码编写，开始重定向流水线上板代码的调试。
第十天	完成重定向流水线上板代码的调试，通过前仿真，检查通过。

5 设计总结与心得

5.1 课设总结

流水 CPU 是现代 CPU 的基础也是较为核心的技术之一,通过使用 logisim 和 fpga 平台完成课程设计,加强了对流水 CPU 体系结构的理解。做了如下几点工作:

- 1) 设计了单周期 CPU 上板模块架构以及数据通路,设计了流水线的部件划分、数据通路、流水接口,设计了气泡流水线的气泡产生机制,设计了重定向流水线数据转发的机制,设计了多重中断、流水中断的中断机制。设计了 fpga 平台重定向流水线的模块架构。
- 2) 实现了单周期 cpu 的 verilog HDL 描述,实现了气泡流水线、理想流水线、重定向流水线,单周期多级中断,流水线中断的 logisim 描述,实现了重定向流水线的 verilog HDL 描述。
- 3) 完成了单周期 cpu 的上 fpga 开发板测试,完成了气泡流水线、理想流水线、重定向流水线,单周期多级中断,流水线中断的测试与检查,完成了重定向流水线上板的前仿真和测试检查。

5.2 课设心得

为期两周的课设做得比较累也较为有成就感。体会与收获有以下几点

1. 先设计,后实现。在做气泡流水线的时候,其实没有想太多就开始数据通路的重构了,一开始将 NPC 放在 ID 段结果后面发现这样做很别扭又进行了大幅度的改动。
2. 交流讨论确定最优方案。还是为了开始做之前把坑都摸清楚,还是应该要和其它小组成员先把方案讨论一遍,即使别人还没开始做,在讲的过程当中有些问题就浮现出来了,比如说做流水中断时流水中断放在哪一段,各个人有不同的做法。交流一下就好比较优劣了。
3. 代码 review 和统一代码格式,虽然在分工完成的 CPU 上板这个实验里面统一代码格式比较难做到,但是代码 review 比较重要,建议规定自己写的代码自己写完的部分给别人测,别人一方面要读懂你的代码以检查逻辑错误,另

华中科技大学课程设计报告

外一方面也方便互相熟悉形成统一的代码风格。

4. 版本管理。这真是血和泪的教训，每天提交一个版本，过一个检查点提交一个版本，尽量用云存储，我就是被坏了的 U 盘坑掉了动态分支预测的。

同时有如下一些建议

1. 重定向流水线上板代码量远远高于单周期上板，建议重定向上板改为小组分工，从第二周开始做。这样子，一方面减少了重定向上板的工作量，另一方面，在商量重定向上板分工的时候会促进小组内之前实现的各个版本之间的交流比较，从而选出最优方案，也提高着小组的整体完成度。
2. 强制版本控制，要求每天提交一个版本。

参考文献

- [1] DAVID A. PATTERSON(美). 计算机组成与设计硬件/软件接口(原书第 4 版). 北京: 机械工业出版社.
- [2] David Money Harris(美). 数字设计和计算机体系结构(第二版). 机械工业出版社
- [3] 秦磊华, 吴非, 莫正坤. 计算机组成原理. 北京: 清华大学出版社, 2011 年.
- [4] 袁春风编著. 计算机组成与系统结构. 北京: 清华大学出版社, 2011 年.
- [5] 张晨曦, 王志英. 计算机系统结构. 高等教育出版社, 2008 年.

• 指导教师评定意见 •

一、原创性声明

本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。

特此声明！

作者签字：

李沐辰